Conf-9410314--3

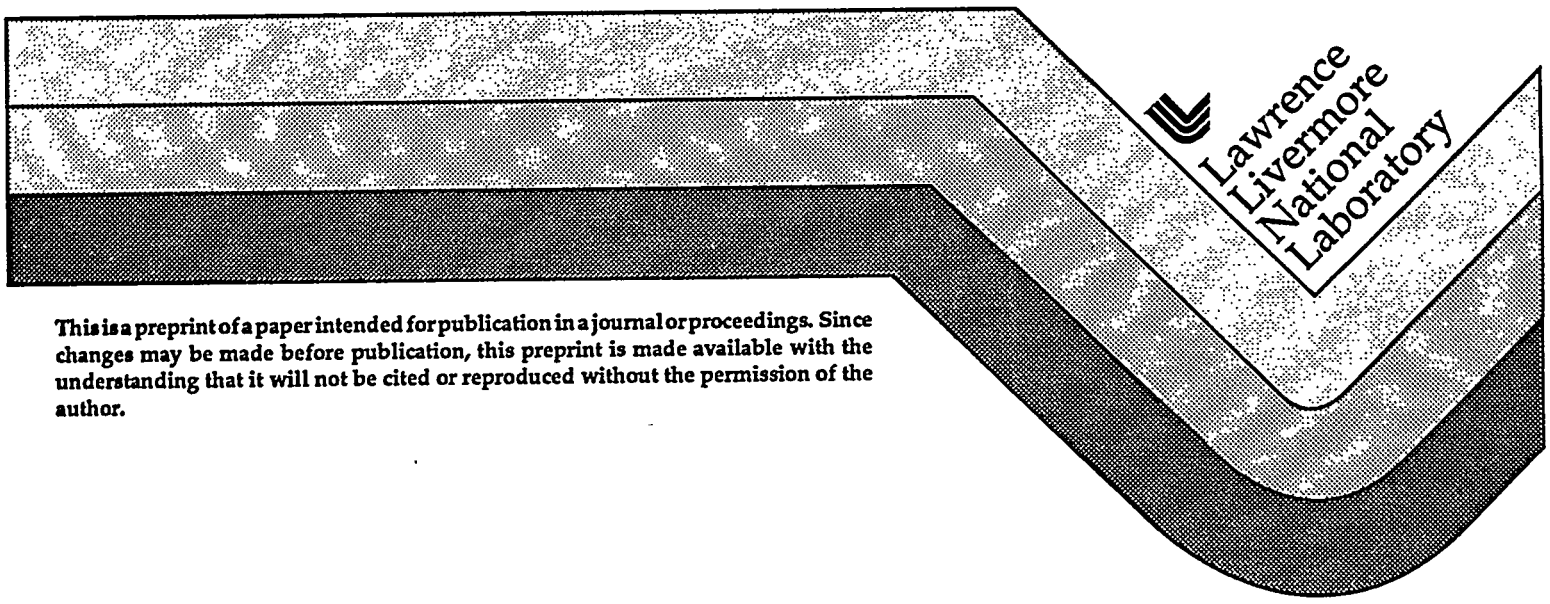
UCRL-JC-119826
PREPRINT

The Development of a Distributed Computing Environment for the Design and Modeling of Plasma Spectroscopy Experiments

J.K. Nash, J.M. Salter, W.G. Eme, and R.W. Lee

This paper was prepared for submittal to the
6th International Workshop on Radiative Properties of Hot, Dense Matter
Sarasota, FL
October 31 - November 4, 1994

October 1994



THE DEVELOPMENT OF A DISTRIBUTED COMPUTING ENVIRONMENT FOR THE DESIGN AND MODELING OF PLASMA SPECTROSCOPY EXPERIMENTS

J. K. NASH,† J. M. SALTER,‡ W. G. EME,† and R. W. LEE†

†Lawrence Livermore National Laboratory, L-58, P.O. Box 808, Livermore, CA 94551, U.S.A., and

‡South Gosforth Computer Systems, Ltd., 21 Broadwell Court, Castle Dene South Gosforth, Newcastle Upon Tyne, NE3 1YS, U.K.

Abstract—The design and analysis of plasma spectroscopy experiments can be significantly complicated by relatively routine computational tasks arising from the massive amount of data encountered in the experimental design and analysis stages of the work. Difficulties in obtaining, computing, manipulating, and visualizing the information represent not simply an issue of convenience – they have a very real limiting effect on the final quality of the data and on the potential for arriving at meaningful conclusions regarding an experiment. We describe ongoing work in developing a portable UNIX environment shell with the goal of simplifying and enabling these activities for the plasma-modeling community. Applications to the construction of atomic kinetics models and to the analysis of x-ray transmission spectroscopy will be shown.

1. INTRODUCTION

Historically, the role of computers in physics research has been viewed as a fast and sophisticated calculator. More recently it has been recognized that computers can be of more general utility in all phases of work including experimental design, data preparation, solution of equations, data reduction, model comparisons, and even the filing and communication of results. Even within the traditional role of numerical calculations, we find an increasing dependence on computers due to the vast quantities of data which enter into the work, both as input and as output. A consideration of recent work in plasma spectroscopy modeling shows this dependence throughout the field, including modeling of ionization balance,[1] transmission spectrum temperature diagnostics,[2] Stark-profile density diagnostics,[3] and other more general spectral modeling activities.[4,5,6] In additional to the "number intensive" aspect of this work, difficulties in computational modeling arise from several less-obvious sources:

- First, plasma modeling is a highly multi-disciplinary field. Analyzing a single experiment may require significant contributions in the varied areas of hydrodynamics, population kinetics, atomic physics, collision physics, spectral line broadening, *etc.* Typically, these processes are computed by separate computer codes, each with different data requirements and formats. Transferring results from one code to another can, in itself, pose problems.

- Also, we find an increasing trend toward computations in heterogeneous environments composed of distributed workstations. Both the differences in the workstations, and their distribution over a network bring new problems to the sharing of data and codes.

- The highly-collaborative nature of the work can present challenges. Increasingly, we find that data and codes need to be shared, and thus need to be understandable by people other than the code authors.

Aside from outright misunderstandings and code misuse, this can lead to "author bottleneck" where obtaining a result, or a calculation with special requirements, is dependent on the availability of the code author.[7]

- Necessary reference data is generally limited, difficult to obtain and unstandardized in format. Data centers are, by and large, unprepared for general, unregistered user access.

- Theoretical datasets can be massive, difficult to obtain, and of mixed format and quality. Certain databases are available through direct network (*i.e.* anonymous ftp or World Wide Web) connections[8-10] but the location of the archive, the specialized content, and uncertain assumptions inherent in the data can impede effective use.

These factors combine to adversely affect the quality of scientific work. The cumbersome nature of working with massive datasets tends to limit the number of studies and inhibits critical analysis of the work. As a consequence, results from such studies can be inconclusive with questions of completeness and procedure not uncommon.

At present, the plasma modeling community depends heavily on the limited and often overwhelmed resources of specialists to obtain datasets of atomic, opacity, or ionization kinetics information. Such limited access to the data has several harmful effects on the field: overall activity is restricted; access to the field by entry-level scientists becomes difficult (small academic institutions are particularly affected); peer review is restrained, as the overlap of skill in the field is limited; and studies employing obsolete methods and data sources are still accepted as representative work. Therefore we see that the size of the computational problem and the current expert-based approach inhibits broad understanding.

To improve this situation, we have undertaken the development of a highly-interactive, distributed computing environment dedicated to the design and modeling of plasma spectroscopy experiments. The work will,

in particular, focus on making available and *useful* the datasets required for research in plasma spectroscopy. The utility of this project is in resolving the difficulties presented above. The timing of this work is opportune: in recent years, we have seen the wide availability of low-cost, highly-capable scientific workstations so that former "supercomputer" capability is now available to individual researchers throughout the world. Also, the increasing access to large computing networks (*e.g.* the Internet) is providing a great stimulus to collaborative activities. We feel that, technically, it is appropriate to undertake this work at this time, and that, to minimize barriers to understanding, it is necessary that we do so.

The *principia* for the project follow from the goal of providing a comprehensive capability in a user-oriented setting. The concept is to provide a complete set of portable, computational tools that work within the framework of a modest workstation environment. The tools must provide a range of capability as required to work in the field, as for example a capability for assembling and manipulating atomic kinetics models. The individual tools – obtained from the working community – and the environment itself, must represent the state of the art in their particular areas. As with other software, to be acceptable, a high degree of intuitiveness and simplicity is essential. In addition, the capability we provide must not artificially dictate a user's solutions. That is, we must provide a general and flexible capability.

These objectives impose a set of technical specifications for the environment. In brief, this environment will consist of an interactive C programming language interpreter; an integrated interface to local and remote databases; a graphical user interface, and access to portable, structured binary data files. The environment will support distributed, cooperating computing processes and provides for linkage with private user-developed physics codes and function libraries. Each of these aspects will be described in some detail in the sections which follow. For now, we point out the overall requirements that, to ensure acceptance and utility, the system must be intuitive, portable, standards-based, readily available, and designed in a manner consistent with its intended use in plasma spectroscopy modeling.

The design principles adopted here have, in part, been based on observations of the degree of success obtained by related developments in such diverse areas as image processing,[11] astrophysics,[12] and plasma simulation.[13,14] The inherent capabilities of each of these efforts are significant. Nonetheless their acceptance has, in some cases, suffered, which we believe is correlated to the extent that they deviate from the above principles. We will comment briefly on these points.

Typical usage patterns of scientific computer systems show periods of intense use alternating with relatively quiet periods. Thus it is unreasonable to expect anyone to remember unusual or highly-specific aspects of use. This leads to the requirements of intuitiveness and standards-based design. As regards the interpreter, a non-standard language or one which is partially standard, can lead to a limited acceptance by the intended users. The requirement for portability follows from the fact that every computing platform has its idiosyncrasies. The best hope for wide-spread use is to depend on supported, generally-accepted, portable "middleware" for the graphics and windowing systems, the binary file systems, and for process communications. We find that, no matter the size of the institution, the cost and licensing aspects of software are significant issues. We thus are avoiding dependence on commercial and restricted-use products. Finally, it is important to keep the needs of the plasma modeling community foremost. Deviating from the needs of the community will inevitably lead to a loss of acceptance.

It is appropriate to note that this paper describes a work in progress. This endeavor represents a significant effort, and is not fully complete at this time. Some example applications will be presented in the final sections which indicate the current status of the work.

2. THE COMPUTATIONAL SHELL

Of primary importance is that the computational environment be interactive. It is instructive to note that at present a great deal of data reduction and analysis is carried out with general purpose commercial plotting packages, several of which exist for the different common

computers in use (*e.g.* PC's, Macintoshes, UNIX workstations). Interactive systems are extremely useful for the exploratory review of data and modeling results and for carrying out tasks where the final criterion is a non-quantitative visual measure, as in data smoothing, background subtraction, and spectrum matching. Simple plotting is usually inadequate for these purposes. It is generally necessary to have some level of data manipulation or programming capability. We therefore envision the user environment as a shell providing an embedded, interpretive C language processor in conjunction with a set of capabilities specifically addressing the field of plasma spectroscopy. The interactive language interpreter is an essential key to achieving this goal as it provides the framework for implementing many of the features we require. In particular, it provides a capability for data manipulation, a vehicle for code development, coordinates access to databases, provides the graphics capabilities, supports a sophisticated structured binary file data access, allows a transparent linkage to user-developed applications and routines, and enables a graphical interface for a user's routines. These topics will be discussed further.

In deciding upon the choice of language, we had to ensure that it provided the necessary flexibility and functionality, but did not require an unreasonable familiarity on the part of the user. As noted, since the shell will likely be used on an intermittent basis we must choose a language which is already commonly known, is standardized, widely-available, and which is compatible with current developments in workstations and computer science. Only two choices present themselves: FORTRAN and the C language. While FORTRAN is undeniably more widespread in scientific usage, we find the utility of C's data pointers and structures, and its underlying relationship with the UNIX operating system a convincing choice. We note that the C language can be employed in a simple fashion quite like FORTRAN so that casual users need not be overly concerned. In addition, we support a transparent linkage between the shell and a user's externally compiled code (in *either* FORTRAN or C) which allows one to continue working in FORTRAN and to use pre-existing code. We consider it unacceptable to provide a language which is a bit like

FORTRAN, a bit like C, and a bit of imagination as this would impede its utility and acceptability. In sum, we provide an interpretive (ANSI) C language capability that supports the standard data types (*e.g.* arrays, pointers, and structures), control loops (for, while, if/then, switch/case), function calls, and access to the standard C libraries. As the ANSI standard is being followed, any convenient textbook on the language serves as a reference. The additional capabilities which we provide, in particular the data manipulation and graphics capabilities, are by means of standard function call conventions. A further benefit of following the language standard is that a piece of code developed in the shell may be directly ported to compiled code. This code may be used either independently or accessed from within the shell via linkage. The usual execution speed penalty for interpretive code is thus avoided.

In dealing with massive amounts of data, it is often convenient to be able to manipulate the information as arrays, rather than as individual values. Aside from the simplicity and clarity that this allows, one can obtain a significant speedup in calculations since the interpreter is invoked only once for the full array. This has been previously recognized and full computer languages, such as the APL language, have been designed toward this end. More recent developments oriented toward scientific use[14, 15] have also followed this approach. For reasons of standardization mentioned above, we have chosen not to implement array-based manipulations as part of the language syntax, but to provide this via standard function calls. These functions can automatically handle variables at different levels of dimensionality (scalar or multi-dimensional array) or type (*i.e.* real or integer). This frees the user from constantly passing along the variable's dimensions as arguments, or calling a set of different routines depending on its type. This is especially useful when dealing with data from a database. In such cases, the amount of data returned from a request varies, depending on the specified constraints or the completeness of the database. Also, as there are fewer parameters to be passed between routines, the reliability of the user's work is increased. A large number of these routines have been provided and are used to advantage in the atomic kinetics modeling application. The functional

capabilities include simple binary operations, such as arithmetic, relational, and logical, as well as more complex operations such as indexing, subset selection, and array catenation. The fundamental argument in support of array-based functionality is that it is insufficient to provide ready access to databased information if it remains cumbersome to work with the data *after* it is obtained.

3. DATABASES

We use the term "database" to describe essentially any logically organized collection of reference information (experimental or theoretical) employed in a physics research problem. This rather loose definition is appropriate since it characterizes the highly varied approach used for data archiving in the physical sciences. Databased information tends to be data which is used repeatedly, which is difficult to compute and/or collect in an organized fashion, or which requires special expertise to obtain. Aside from providing a repository for the information, a database may provide convenient methods for accessing subsets of the data. However, this is much more true in commercial database implementations than in the field of physics.

A user's application drives the data requirements so that, in the present area of work, we tend to think of fundamental atomic properties, equation-of-state data and experimental measurements as information suitable for databasing. One envisions accessing an atomic physics database for such purposes as constructing a comprehensive atomic kinetics "model", or for more specific applications involving compound and complex atomic processes. Examples include calculations of line shapes for multi-electron emitters, dielectronic recombination coefficients, or the synthesis of beam-excited electron emission spectra.

We believe the utility of databasing to be much more general than this. For example, the output of hydrodynamics codes can be profitably archived. This information tends to be massive and can be used for a variety of post-processing applications. The input to such codes is equally appropriate. One typical input might be the spectral definition and

history of the drive pulse available from a specific laser facility. This is usually not a strictly experimental quantity but is a construct prepared with some effort. A library of such quantities characterizing the different facilities would be of general interest. Further, the startup file used to specify a hydrodynamics calculation could be usefully archived, particularly if tagged by the salient parameters. These startup files can be tiresome to construct due to the large number of sometimes obscure defining options typically required. This, however, would not be particularly interesting to researchers beyond the author's working group. In sum the output from any calculation, computer or hand-constructed, is appropriate for archiving if it has further utility either to the author or to other workers.

As this suggests, we propose a rather general usage of databasing. We must now consider what kind of technology is appropriate for implementing this capability. The applications described above show that database use can be grouped according to whether the data is only of local interest, or is useful to the community as a whole. Also, it is not always necessary that a database be relational. A relational database is designed to allow very general access to the information, though at an overhead cost of access time, storage space, and in the design of the database itself. A relational database would not be necessary for a situation where general selectivity was not required. As an example of a non-relational database application, consider that a hydrodynamics calculation requires access to the full extent of an equation-of-state database. The only required selection criteria in this case is the atomic composition of the material of interest.

Given the varied use of databases and the need to support existing relational databases and text file archives, we recognize that several different database implementations will need to be supported. In addition, some highly-idiosyncratic, or limited-interest databases are distributed privately and consist of a data file together with a set of access routines. In this case it is probably best to use the supplied function call interface. In the shell, we implement an ideal which is to access databases in a manner that is independent of the type of the database. In this

approach a library of access functions has been developed which confine the database-specific portion of the access to a single routine which specifies the data to be read.

The overall process of accessing a database requires, at minimum, a means of initiating a connection to a database server, selecting the database to access (*e.g.* argon or aluminum), requesting subsets of data, and terminating the connection. It is primarily in specifying the data request that database-type dependencies occur. The access routines are designed to be simple to use and resemble working with standard UNIX files. All operations utilize a DATABASE descriptor in the same spirit as the C language FILE descriptor, or the FORTRAN logical I/O unit number. The advantages of using the shell database routines are many: space for the returned data is automatically allocated, the result is returned as a data structure which may be easily passed to other routines, the returned data structure is independent of the actual database type, a frequently accessed database may trivially be re-written as a database of a different type (for example a structured binary file database, with a significant speedup in access time), it requires little change to read a database of a different type, and database access is quite straightforward using the shell routines.

The types of data archives which are presently accessible with the shell interface are briefly described below. It is relatively simple to support additional types by providing new server routines. Server routines are described further in the following section.

relational databases

An atomic physics database is representative of a database type which has general utility throughout the field, can contain a massive amount of information, and for which selective, relational, data access is of benefit. Accordingly we have developed an atomic physics database on a UNIX workstation using a standard, commercial relational database product. Such products are available from several vendors – the choice is arbitrary since the widely-accepted Standardized Query Language, SQL, is used in all transactions. A SQL-compliant database can be viewed as a set of

tables, where each table has a number of columns. The columns are themselves identified by name. Each row in a table is one entry in the database. The atomic physics database is organized such that the different tables have information on the physical states, basis states, radiative transitions, collisional cross sections, *etc.* Additional tables describe the codes used to generate the stored data and other aspects of the calculations.

Note that as the quantities stored in a database can have arbitrary names, it is essential to provide a means for obtaining the schema (organizational structure) for the database. Since this can be true of other database types, we have provided an additional shell access function to return this information.

The specification of the data request is in the form of a SQL "select" command, which is of the following general form:

```
select x, y, z from table1, table2 where constraint1 and
constraint2 or constraint3.
```

The selected quantities `x, y, z` are the names of the desired quantities, exactly as appearing in the schema of the database. Examples from the atomic database are: `state_energy, parity, totalJ`. The "from" field indicates which tables are to be searched. The final portion of the command presents search constraints. In the absense of constraints the database will return all entries of `x, y, z` data in the database. For this database, we generally have constraints to apply, such as to return only those results where the atomic ion has 5 electrons, or where an x-ray line position is within some bounds. As required, quite complex and exacting constraints can be constructed. In particular, the constraints of one request can be included with another in order to form correlated requests. This is advantageous for the purposes of developing atomic kinetics models and will be discussed further in Section 7.

structured binary file databases

In contrast to the example of an atomic physics database, the output of a user's code containing results for x-ray absorption coefficients or hydrodynamic simulations are best stored in a different type of database. In such cases, the selectivity of a relational database is not required, but rapid access to the data is. Application specific databases such as these may be implemented using portable, structured binary files designed for use as scientific data management systems.[16] A simplified interface to two of the most general systems, netCDF,[17] and PDBLib,[18] is provided as part of the shell and in UNIX library form for use in stand-alone applications. These file systems offer compelling advantages over native binary access. As a databasing scheme, they also offer specific advantages over the use of a relational database. In brief they provide fast access to the data, allow both read and write access to more complex data structures than provided by SQL, have no overhead in defining the database, and are widely used by research groups and many commercial application programs. Of additional importance, they are institutionally supported yet reside in the public domain.

While these file systems do not provide the access authentication capability or general relational access provided by commercial relational databases, these are also among their advantages. The complexity of administering a database system and, as required, *bypassing* the authentication system are often considerable nuisances for the owner of a database. Since structured binary files do not natively support SQL commands, only a somewhat abbreviated SQL syntax is made available by the shell. For example, it is not possible to impose constraints in the "select" command – the full variable is retrieved. An example of a database composed of PDB files is presented in Section 8.

text file databases

The most common form of archiving is tabular data in simple text files. Such files may be accessed from within the shell by the usual C language I/O libraries. Due to their frequent use as a "database" in the form of tabular arrays, we additionally provide access via the database library.

These are the same functions as used to access any of the other database types. The syntax for indicating a selection of a text file reduces to specifying the lines and columns of interest. As a convenience, delimiters may be specified to eliminate the need for precise formatting statements. For such file types, the emphasis is on simplifying the access to the data.

ALADDIN databases

Within the plasma physics community there exists a standardized file format for representing atomic physics transition data. This format, the ALADDIN data format,[19] has been developed under the auspices of the International Atomic Energy Agency (IAEA) and consists of both a data formatting standard for a diverse set of processes and a set of data access subroutines. The term "ALADDIN" is also used to refer to a specific collection of atomic rate data maintained by the IAEA (in ALADDIN format).[9] The data files themselves are standard text files. This is an example of a privately distributed database which could simply be linked to the shell and used according to its documentation. Due to the international charter of the system, and its possible wide-spread use, we have chosen to support access to ALADDIN data files via the shell database library. As in previous cases it is only in the syntax of the data request that this type of database differs from the others. The ALADDIN syntax is quite unusual and we have not attempted to cast it into a SQL-like form. As a consequence it remains necessary for the user to specify his request in the ALADDIN form. In other respects access to these databases is identical to the previous cases.

4. DISTRIBUTED DATA ACCESS

Access to distributed data arises in two contexts: access to remote databases, and access to files residing on other machines as a result of calculations carried out there. In the latter case, the data is of a temporary nature. The goal is simply to bring the results of the calculation back for use by the shell. Remote databases are data archives of a more permanent nature which generally have some specialized data access procedure, for

example using SQL commands. We have implemented an approach where both sets of data are accessed in a similar fashion.

Interest in distributed, or remote, databases is not a recent development due to the increased use of data networks. In many cases databases which are developed for independent reasons in varied locations are later seen to be of great value if made generally accessible. So, one is attempting to facilitate access to data which is already distributed. This is the case in databases of atomic energy levels and processes (a good summary of atomic data archives throughout the world is given in Ref. 20). A difficulty with such a disordered development that there is very little commonality in the software implementation of these databases. In general, the access software is database specific, the interface is by command line only (*i.e.* there is no callable function interface), the returned data is usually very specific and limited (*e.g.* eigenstate properties may not be available), and the "database" may simply be a file directory. Additionally, most of these databases are not on-line.

Given the varied nature and location of these archives, we have chosen the PVM message passing interface[21] as the most reliable and capable means of communication. PVM stands for "parallel virtual machine" and is intended specifically for facilitating the management of processes and data on a heterogeneous, distributed network of cooperating computers. The "virtual" machine is defined by enrolling actual machines. The enrollment process consists merely of specifying the participating computers by network address. The processes to be connected may be running either locally or over a network. PVM also has the capability of "spawning" (*i.e.* starting), processes on any machine enrolled in the virtual machine. This is invoked through the agency of a PVM server daemon. In the present discussion, we connect to distributed databases by requesting PVM to spawn a remote server process which can communicate with the database. As the databases have highly specific interfaces, the server process must be of an appropriate type in each case. To the extent possible, these server processes speak in standard SQL on the network side and in whatever fashion is required on the database side. This approach restricts the idiosyncrasies of specific databases to just the server code. As a

consequence, new database types may be supported with no changes in the shell. Since PVM transparently integrates all data resources in the virtual machine the shell need never concern itself with the actual physical location of the requested data, aside from initially specifying its address.

Database server routines presently exist for accessing SQL-compliant databases, structured binary file databases, ALADDIN format data files, and simple text data files. The process begins with the shell requesting that a server process of a specific type be spawned on a specified machine. The server process created by this request then initiates a connection back to the shell, after which the shell provides the name of the database to be connected, or the complete file path of the file to be accessed. The server then connects to the database and enters into a sequence of transactions with the shell. All the steps to establish a connection with a database are transparent to the user and are invoked when the database "open" function is called. A database transaction consists of the issuance of a "select" operation upon the database, after which the server process assembles the result and transmits it back to the shell. When a database "close" function is called, the server receives an instruction to close its connection to the database and quit. In the case of simple file transfer between machines (for example the input or output file for a calculation on a remote machine) the server process has a similar dialog with the shell, but simply transfers the file and terminates.

Dealing with a remote, general-access database presents new complications regarding access authentication, database availability, multiple- versus single-use licenses, and whether the requesting process, usually the shell, should block – that is, wait – for the data request to complete. An additional consideration is the question of locating a distributed database. When requesting that a database be opened, it is necessary to know the network address of the archive. At present the user specifically provides this information. A preferred approach is to establish a network directory server to translate published database names into a complete specification for accessing the database. This would include the machine address, the type of server required, and as necessary, the pathname to the data.

5. PROCESS COMMUNICATION

A principle goal of the developers of the PVM library was to enable calculations to be carried out by the cooperation of multiple distributed processes on a varied set of machine architectures. Multiple processes can cooperate by the parallel execution of a specific calculation, *i.e.* parallel processing, or by coordinating an overall task such that different processes carry out specific portions of the work. This latter approach, termed distributed processing, is the model used by the shell. We are not, in fact, specifically seeking a distributed processing capability. In general a user will be running the shell and perhaps one or two specific application programs. In many cases, these processes will all exist on the same machine. The use of other machines will most likely occur in the context of a request to access a remote database. Distributed data processing may also be attractive if a significant amount of data is to be obtained from a remote site. In general, we prefer to carry out as much of a calculation as possible at the site where the data actually resides so as to avoid unnecessary data transmission. This would not be possible in the case of a public-access database server where the only permitted function is data retrieval, but would be possible on a machine where the user has more general access permissions. Another scenario for distributed processing is to call upon a high-performance workstation or supercomputer to carry out an intensive numerical calculation. The results of the calculation would then be returned to the shell on the local machine for analysis, archiving, or for input to a further calculation.

All these activities can be handled within the PVM framework. This library can handle communications either locally or across the network and provides for data format conversions between machines of a different type. PVM has a high level of institutional support and is widely-used by scientific research groups.

We support the use of distributed processing by allowing the user to spawn new processes, on any cooperating machine, at will from within the interpreter shell. When a new process is spawned, the user is returned a PROCESS pointer which, like a FILE or DATABASE pointer, carries all

the information needed by other shell routines to communicate with the task. Communications functions are available which allow a user to implement data transfer and process synchronization between the shell and the new process. One application of process synchronization is to coordinate a hydrodynamics code and an atomic kinetics solver so that as the hydrodynamics steps through time, the kinetics solver receives updated spatial state properties. The kinetics solver could then compute new level populations and pass them back. Yet another code could then use these data to compute x-ray emission spectra as a function of time. A user can thus create new physics simulation capabilities by coordinating existing packages. The different physics packages are completely isolated (perhaps even on different machines), except for the information specifically transferred. As a consequence, modifications in one code are unable to cause unexpected effects in the other. This is an extreme example of "data hiding" which is strongly encouraged as code development gets large or multiple authors are involved.[22]

6. GRAPHICS AND GRAPHICAL INTERFACES

The shell graphics capability is intended to be compatible with, and used in a fashion complementary to the large number of data plotting packages already in wide use in scientific settings. Sophisticated, commercial multi-dimensional visualization software is readily available to those requiring it. Additionally, everyone will, by natural selection, continue to use the programs they find most comfortable. However, as it is so much easier to comprehend trends and relationships visually, it is of great importance that provision is made for a convenient and reasonable data plotting capability integral to the shell. Further, there are special graphing requirements in the plasma spectroscopy, notably Grotrian diagrams, which are not generally found in other software packages and which we will provide.

As with other aspects of this shell, we restrict ourselves to the use of standards. In this case we have adopted the X Window System[23] with the Motif interface standard[24] for all graphical work. This system is suitable for both data plotting and for the development of graphical interfaces. In

addition to the strength and flexibility inherent in X Windows, another advantage is that a great number of compatible extensions exist in the public domain. In particular we have employed the plotting widgets available in the Athena widget set.[25] A good deal of capability is obtained this way. As a standard part of the shell we provide a plot capability for: multiple curves, contour plots, double x and y axes, linear and logarithmic scaling, arbitrary text placement, and creation of PostScript™ display files. There is editing control over all aspects of the display, including tic marks, font size and style, curve patterns and color, axis scales, and other features commonly expected. The creation of the plots and setting the desired display properties may be done by function calls, or by use of a graphical plot editor interface.

The plot widgets also provide a "callback" capability, which allows the user to specify a function to be called when a particular event occurs. The events which are supported are: motion of the cursor within the plot window, the selection of points or objects within the plot by use of a mouse, and the specification of rectangular regions within the plot by a click-drag sequence. These callbacks are easily employed by the user to select plotted items for modification, to choose an (x, y) coordinate pair within a plot, to extract a set of (x, y) coordinates as a path, and to interrogate the system about the properties of selected items. The utility of callbacks in an actual data analysis setting is illustrated in Section 8.

The close relationship of the plot package to the shell interpreter allows selected items, such as a displayed curve or a set of coordinate pairs to be created as new variables within the interpreter for further use (perhaps as input to some calculation). The connection to the interpreter also works in the opposite direction, so that if a data variable in the interpreter is plotted, any modification of the data will cause a re-draw of the curve in the plot. This allows the interpreter to carry out data reduction activities on a plotted curve with a continuously updated display of the results. Note also that this allows a plot to be tied to the updated results from a spawned simulation calculation. As the simulation advances it sends the updated information, *e.g.* zonal state properties, back to the shell which then immediately updates the plot display. We refer to this close,

interactive relationship of the interpreter, plots, processes and the user, as *active* graphics.

A graphical user interface (GUI), while often criticized as an unnecessary decoration, has particular utility in the context of a database or file browser. Since the schema of a database may be difficult to remember, it is very useful to get a quick display of the definitions of the database tables. It is also useful to view files before reading them to ensure that the correct file has been selected. A GUI is not essential, but it does make these tasks simpler and more direct. Another advantage of a GUI is, if properly implemented, the simplicity that it provides to the user for setting parameter values in an application interface (for a plotter, a hydrocode *etc.*) that would otherwise require a complex guidebook and an arcane function call interface. A GUI also allows much more rapid changes as it can largely bypass a keyboard-based interface. Finally, a GUI allows a user to readily identify choices. For example, in performing an instrumental broadening of a synthesized spectrum, the choices for setting the resolving power (*e.g.* fixed FWHM, $\lambda/\Delta\lambda$, Bragg law variation) are immediately apparent. Aside from the convenience factor, there is less opportunity for error when the interface is made more intuitive.

It is a challenging undertaking to provide a user-oriented package for GUI development. We have chosen to provide a limited capability, yet one sufficient to allow a user to specify text display and input boxes, radio and check boxes, slider controls, two-dimensional (x, y) point selection boxes, scrollable lists, and dialog boxes. These are not particularly difficult to implement as they are provided as part of the X/Motif windowing system itself. Since standard X Windows and Motif library calls are used, an ambitious user is free to add additional capabilities. Such modifications are very welcome. The authors would be interested in user contributions which could benefit the overall community.

7. EXAMPLE I: ATOMIC KINETICS MODELS

The study of high-temperature, highly-ionized plasma sources frequently requires a consideration of the non-LTE character of the plasma. In

particular, the spectroscopic modeling of laser-produced plasmas, especially for such sensitive features as emission line ratios or laser gain estimates generally requires the use of comprehensive, highly-accurate numerical models of the atomic state. These "atomic models" can vary extensively in the detail of the electronic energy level structure and the physical processes which they include. It is quite common, though, for such models to include a multi-ion representation of an atom with a distribution of energy levels within each charge state. Quite frequently a good description of multiply-excited states above the lowest continuum will also be required. Transitions between the atomic states defining a model may include a variety of physical processes. It is usual to include the excitation and ionization contributions from radiative processes, electron collisions and autoionization transitions. Multi-step processes, such as dielectronic recombination or resonant excitation-autoionization are also often included, as may heavy ion and charge-exchange collisions. These latter processes are especially important in studies of magnetically confined plasmas. An atomic kinetics model is also subject to various possible approximations to reduce the number of states to a more manageable size, hopefully without adversely affecting the physical behavior under study.

The energy separation of atomic states depends on the details of the atom but all atoms show a hydrogenic trend of increasing density of states near the ionization limit. The decreasing level separation and the increasingly collisional nature of these highly-excited states provides justification for the use of composite, or "average" level descriptions. Rates connected to such states must then be appropriately summed or averaged to ensure the consistency of the reduced model. In addition, it is important to construct these average levels in a physically appropriate manner to retain the relevant kinetic properties. There is no "best" prescription for specifying the averaging process. This remains an art requiring an iterative reduction, or extension, of a model to obtain the desired balance between size, complexity and accuracy.

One approach to balancing model accuracy and complexity with computational efficiency is to develop a sequence of models of graduated

complexity where the one with the least detail is used to model the gross physical properties, for example the charge state distribution. A more detailed model is then run in a post-processing fashion to compute the properties of spectroscopic interest. This is, in fact, the most common scheme, but it frequently suffers in the implementation in that the two models may bear no relationship to each other. In many cases the models differ in the values of the physical properties (energy level values, rates), the rate processes included, and qualitatively in the manner in which the levels are connected.

The task of constructing and executing averages over an atomic model possessing many hundreds of atomic levels and tens of thousands of rates is sufficiently complex that very few researchers have this capability. In addition, the exercise is sufficiently onerous that only limited investigations are made of the sufficiency or accuracy of the reduced model. Specific rates can be evaluated, gross plasma properties can be compared, but a detailed evaluation of the scaling behavior of the kinetic model is the exception.[26]

It is for this reason that we have included a very general capability for constructing and evaluating atomic models as an integral part of the plasma modeling environment. This "application" is written as a program within the shell itself, using the built-in C language interpreter. The interpreted code provides loops over processes and ion stages, makes calls to an atomic physics database for some of its information, keeps track of the many data constituents, and carries out the steps to define and execute a user-specified kinetics model average. The calculation is not limited by the interpretive overhead since many of the routines used in the calculation are written in compiled code and, furthermore, make use of array processing for additional benefit. We will limit the discussion here to qualitative remarks on the model construction process since the primary concern is with capability and methodology. Implementing the model-making capability within the shell offers a number of advantages to the developer: the shell simplifies the tasks of selection of energy levels, physical process selection and comparison, and evaluation and

modification of the model – all of which are best done within an interactive framework.

Using the atomic physics database

It is worth emphasizing that an atomic model is largely defined once the energy levels and the level averaging process have been specified. Considering any particular process type we know that any transition connecting levels in the model should be included. Thus the set of energy levels is the key item. Specifying energy levels to be obtained from the atomic physics database involves the use of constraints in the `select` statement. These constraints can be applied to any of the properties which define the energy level in the database.

For this reason, the database design we employ includes a number of quantities to facilitate the data selection process. For example, a set of atomic states may be chosen from the database on the basis of the principal quantum number of its outermost electron, and on the number of electrons in the state (*i.e.* the charge state) as follows:

```
select state_energy, stwt, level_key from state_table
where (nbound = 4) and (nouter <= 5)
```

More complex requests are equally easily stated:

```
select state_energy, stwt, level_key from state_table
where (nbound = 4) and (nouter = 5) and (nouter_next = 3)
and (parity = -1) and (code = "mcdf")
```

This request will return data on the odd-parity beryllium-like states of the configurations: $1s^2 3\ell 5\ell'$, $1s^1 2\ell^1 3\ell 5\ell'$, which have been computed with the MCDF code.

The quantity "`level_key`" in the above request is a database-wide unique level identifier assigned to each physical energy state as it is added to the database. An archived transition between two levels also retains the corresponding level identifiers, under the names "`levkey_lower`" and

"levkey_upper" in the rate table. It is thus possible to establish an unambiguous assignment of transition rates between any two levels. By demanding a match between these keys in an SQL "select" request on transition rates, we can be assured that the resulting rates connect the appropriate levels. More generally, the full set of constraints used to select out the energy levels can be simply appended to the transition rate request in conjunction with the level_key constraint. For example consider the following request for energy levels with a set of constraints on the levels:

```
select state_energy, stwt, level_key from state_table
where {Level_constraints}
```

To get all radiative rates connecting these levels, we simply request:

```
select rate, levkey_lower, levkey_upper from
rad_rate_table, state_table where {Level_constraints} and
(levkey_lower = level_key)
```

Further constraints on the rates to be returned, such as those within a wavelength window, or those with a value exceeding 10^{10} sec^{-1}, may be similarly appended to the request. We see then that a SQL-compliant relational database is of great utility in selecting and correlating information of a varied sort. All the difficulties of searching and matching conditions are enforced by the database search engine. Other SQL keywords exist for ordering the results, counting the results, selecting extrema, *etc.*

To obtain this capability requires the existence of codes to compute the quantities of interest, and which provide output that can be installed into the database. This is the case for the (highly-ionized) atomic physics database where a family of atomic properties codes based on the MCDF atomic structure code has been in use for many years.[27-30] These codes have been extensively employed in the calculation of massive sets of atomic level and rate data.

Given that we can extract energy states and connecting transitions from the database, we now consider methods for modifying the model level

structure, either to add levels, delete levels, or to obtain average energy levels of the sort described earlier. The criteria for averaging levels together varies greatly from researcher to researcher, and from problem to problem. As a consequence, a very general capability is required in order to obtain a utility of wide-spread applicability. A general approach to level deletion and consolidation is obtained through the application of logical operations based on the properties of the atomic states. The method is thus relational in the same fashion as accessing the data from the atomic physics database. Two routines have been provided which "operate" on the energy level data structures. These routines are used to indicate levels to be deleted, and the levels to be averaged together. Level deletion will be addressed first.

Deleting states from the model

All the levels resulting from a database request are set by default to be "active". A level may be deleted simply by marking it as inactive. From a calculational point of view, an inactive level does not exist in the problem. Information relating to the level persists in the data structure, so that the level may be restored if desired, but all operations concerning model construction will regard the level as non-existent. The activity status of a state in the level data structure is specified by a Boolean array. This array may be extracted or replaced by calls to the routines: `GetProperty()` and `SetProperty()`. These are general routines to access properties of "opaque" data structures, *i.e.* structures where the specific internal data organization is not of interest to the user. Opaque data structures, in conjunction with these access routines, are used in many contexts within the shell.

We will present a short example on modifying the level structure of a model. Begin by extracting the array of (active) energy levels:

```
GetProperty(BeStruct, ENERGY, &earray);
```

In the above, `BeStruct` is a data structure of beryllium-like states. The second argument indicates what property to extract. The return variable

earray is an array of floating point data. To determine which energy levels have an excitation energy below 600 eV, we perform a relational operation on the energy array, making use of an array-based function provided by the shell:

```
test1 = le(earray , 600.);
```

The function le() carries out the *less than or equal* comparison of its arguments so that the variable test is now a Boolean array whose values depend on the comparison of the energy values in earray with the criterion of 600 eV. To "delete" the energy levels which fail this criterion, one simply resets the selection array in BeStruct with the array test1:

```
SetProperty(BeStruct, ACTIVE, test1);
```

Energy levels above 600 eV no longer exist in the model! Also, since transitions to inactive levels are ignored, we have, in a single step modified the model in a fully consistent fashion.

Much more complicated criteria can be applied to the deletion process. The set of state energies is only one of many state-related properties which can be extracted by GetProperty(). Any succession of relational tests, based either on the state properties or on a more arbitrary basis (*e.g.* only accept the first 47 states), can be applied to determine which states are to be deleted. A simple extension to the above example is to delete those states with an excitation energy above 600 eV and whose outer electron has an angular momentum greater than 3 (such a consideration might be appropriate when developing an opacity model). Continuing the previous example, the orbital momenta of the outermost electron must be extracted and tested:

```
GetProperty(BeStruct, LOUTER, &larray);

test2  = le(larray , 3);
```

The requirement that both the energy and momenta are satisfied is now imposed:

```
test    = and(test1, test2);
```

and the set of active states is modified:

```
SetProperty(BeStruct, ACTIVE, test);
```

Construction of composite, or average, energy levels

An average level is a single, effective energy level which represents the contribution of a set of levels to the kinetics of the system. The statistical weight of this effective level consists of the sum of its members' weights. The energy value is taken as a simple weighted average of the constituent levels' energies, as weighted by their statistical weights. This prescription does not strictly preserve the atomic partition function but the deviation is small when the constituent levels are closely spaced. Ideally, with an appropriate choice of levels, the average level will preserve both the dynamical and equilibrium distributions of the state populations.

An average level will additionally retain as many of the properties of its constituents as is physically sensible. If all the member levels have the same total momentum, the averaged level will too have that value. The same applies to the electron configuration and parity. The computation of the averaged level's properties is thus straightforward. The pertinent question is to determine which levels can be combined in this fashion without compromising the non-equilibrium character of the full kinetics model – or at least the part leading to the observables. To determine the acceptability of a model, one needs to observe, and compare, different levels of model completeness and averaging. Therefore, the process of averaging must be simple, flexible, and direct. The following approach has been found to be effective.

The function `AverageLevels()` is a low-level routine used to specify states (by level index) for combination into an average level. The usage is:

```
AverageLevels(BeStruct, level_list);
```

The array `level_list` contains a set of level indices, defining the new average level. This array may be of arbitrary length. Internally, the routine computes the properties of the average level and adds it to the model. This level is marked as being an average level, and it retains a record of its constituent levels, each of which remain in the data structure. Keeping the sublevels in the data structure ensures that the iterative development cycle remains general. For example, the model's level structure may at any point be completely redefined since the sublevels are readily restored. The level list may include levels which are themselves already composite levels, in which case their list of member states becomes part of the new one. To average levels 5 and 23 together, we type:

```
int   level_list[] = {5, 23};

AverageLevels(BeStruct, level_list);
```

Transitions between, or connecting to levels which are now part of an average level must be properly summed to preserve the strength of the transition. For a dipole transition, we average over initial states and sum over final states. Since the transitions are processed after the model's level set has been chosen, their averaging can be carried out automatically, according to the current set of energy levels, without any intervention by the user. Accounting for average levels and their effect on transitions is integral to the model-making code. The detailed mechanics need never be directly considered by the user.

The level averaging capability is extended by another function, `AverageGroups()`, which simultaneously carries out a number of averaging operations. The call is

```
AverageGroups(BeStruct, groups, select);
```

The second argument, `groups`, is a integer matrix whose rows define the averaging groups. The number of rows in this matrix must agree with the number of levels in `BeStruct`; an arbitrary number of columns is allowed. This matrix is usually derived from properties of the levels. This

approach automatically results in the correct number of rows. The final argument, `select`, is a Boolean array which indicates which levels are to participate in the averaging process. A zero entry in this array indicates that the correspondng level is *not* to be involved in the averaging process. A pre-declared value of `ALL` is available to indicate that all levels are subject to averaging.

The use of this routine is best indicated by example. If we were to combine all levels into groups characterized by parity, we would expect to obtain just two groups corresponding to the even and odd symmetries. This is done by first obtaining an array containing the parity values of the states, and then using that array to define the groups:

```
GetProperty(BeStruct, PARITY, &parities);

AverageGroups(BeStruct, parities, ALL);
```

The array `parities` is a sequence of -1 and +1 values corresponding to odd and even parities. In this case the array `parities` serves as an N x 1 matrix to define the averaging groups. A group is defined by all levels which have the same value in the array. If entries 1 through 23 are -1 and 24 through 46 are +1, then the first group consists of the first 23 levels; the second group is the second 23 levels. After the averaging, we will have just two levels, corresponding to the averages of these two groups. The plus and minus one values may be scattered throughout the array, the routine will still bring them together into the correct group. A more physically reasonable version of this example might be to apply such a parity grouping only to levels whose energies are above some critical value, for example the ionization potential. This requires that the selection array contain an entry of 1 for every level above this value and a 0 for those below. The same group definitions are used, they are simply ignored for those levels which are not selected. The example is modified as:

```
GetProperty(BeStruct, PARITY, &parities);

GetProperty(BeStruct, ENERGY, &energies);
```

```
test = gt(energies, 600.);

AverageGroups(BeStruct, parities, test);
```

We allow for the second argument to be a matrix (and most frequently it is a matrix) to allow for more general group definitions. To construct a group according to the combined criteria of parity and total state momentum, we extract the parity array and the momentum array and concatenate them to form an N x 2 matrix. The groups are still defined as rows (corresponding to the levels) with identical values. In this more general case the rows must be identical, element by element. Rows which have identical parity and momentum values result in similarly defined groups, and belong to the same composite level. The previous example of groups defined by parity, for levels above 600 eV, will now be extended to groups defined by (parity, momentum), for levels above 600 eV. The concatenation of the parity and momentum arrays is performed by the function, `cat`, as shown in the following:

```
GetProperty(BeStruct, PARITY, &parities);

GetProperty(BeStruct, MOMENTUM, &totalJ);

testgroup = cat(parities, totalJ, COLUMN_WISE);

GetProperty(BeStruct, ENERGY, &energies);

test = gt(energies, 600.);

AverageGroups(BeStruct, testgroup, test);
```

Composite levels are frequently defined as levels with identical configurations. This gives rise to several possibilities since a configuration could be defined by relativistic orbital occupation numbers, non-relativistic occupation numbers, shell occupation numbers, or some combination of these. For this purpose we ignore complications due to multi-configurational basis states. As the occupation numbers are known to the database, by default we extract them and make them part of the level data structure. Like other state properties, they may be extracted:

```
GetProperty(BeStruct, NLJ_CONFIGURATION, &nljconfigs);
```

In the case of N levels whose electrons are all in shells up to principal quantum number $n = 4$, there are 16 occupied relativistic orbitals ($1s_{1/2}$ to $4f_{7/2}$). For this case, the above function call returns the integer matrix, `nljconfigs`, which is of size N x 16. For the same levels, using `NL_CONFIGURATION` to extract the *non*-relativistic configurations would result in an N x 10 array of non-relativistic occupation numbers (*1s* to *4f*). A configuration array so obtained may be used directly to define the groups for averaging:

```
AverageGroups(BeStruct, nljconfigs, ALL);
```

This results in a model with relativistic average configurational levels. As in previous examples, we could modify the group definition by a column-wise concatenation of the state momenta to the array `nljconfigs`, or modify the levels to be affected by changing the selection argument. A frequent variation is to define levels where the outer electron is shell-averaged and the inner, core electrons are averaged by non-relativistic configuration. Within a single ionization stage, this prescription results in sequences of core-excited Rydberg levels. Other functions are available to simplify the construction of composite levels according to such variations as energy binning or thermal bands.

Internal calculation of energy levels

We have thus far discussed energy levels as obtained from an atomic database. It is also possible to construct models whose energy levels are, in whole or in part, supplied by the model-making code. These energy levels are based on an internal calculation of Dirac-Fock-Slater (DFS) single-electron energies.[31] Multi-electron relativistic configuration levels may be constructed wholly with these results. Or, in conjunction with *ab initio* atomic structure calculations, these energies may be used to construct sequences of core-excited Rydberg levels. In such levels, the departure of the running, valence electron from its continuum is determined by the internal DFS calculation. The averaging and deletion processes described previously may be employed regardless of the levels' origin.

Internal calculation of transition processes

The transition data in a model may be obtained from a variety of sources. The default preferred data source is the atomic database. However, a large number of processes may be required in a model and the database may lack the necessary information. In this case the user may specify among a number of secondary data sources. These consist of theoretical or semi-empirical models of the process which can be carried out in-line. A variety of internal calculations are available, depending on the particular process in question: oscillator strengths,[32-34] auto-ionization rates,[35] cross sections for electron collisional excitation,[36-43] heavy-ion excitation,[44] photo-ionization,[45,46] and electron collisional ionization.[47-58] In addition to these, a user may provide his own rate routines in FORTRAN or C code, and have them linked to the shell for use in atomic model construction. The inverse processes for all transitions, including those obtained from the database, are computed by the principle of detailed balance. This occurs in the routines which evaluate and solve the rate matrix, not in the preparation of the atomic model.

The internally available calculations generally provide cross sections or rates appropriate for a single electron interacting with an effective nuclear charge. We employ statistical branching ratios and screening coefficients[59] to apply the cross sections to transitions between more highly resolved levels, or configurations with inner-shell vacancies.

Internal representation of transition processes

Electron-ion collision processes are computed as cross sections – continuous functions of the relative energies of the colliding particles. For the inelastic processes of interest in this work, the usual approximations of isotropy, rapid thermalization among like particles and the extreme difference in mass between the ions and the electrons indicate that in the final kinetics model the transition process need only be represented as rate coefficients, *i.e.* cross sections integrated over an assumed (Maxwellian) electron distribution function. To provide additional capability, by default we leave these processes as cross sections within the models. This allows the user to use non-Maxwellian electron sources or to model electron

scattering processes. For situations where this generality is not required, it is possible to "process" the collision cross sections into the more common rate coefficient form, assuming a Maxwellian electron distribution. This processing results in a parameterized (in temperature) fit to the rate coefficient. The user can thereby eliminate the overhead of computing the rate integral each time the rate matrix is set up.

Energy level identification

New problems arise when attempting to use an approach as general as that presented here. In particular we must consider how to combine the results of atomic rate calculations from different codes. The usual context for this problem is when the atomic structure calculation used to define the energy levels in a kinetics model differs from the atomic structure calculation used in computing the rates which we wish to use. An *ab initio* rate calculation gives results in terms of transitions between levels as defined in its reference atomic structure calculation. Due to different numerical or physical models, the level definitions of different structure calculations may not agree. A mapping must then be determined which can reliably treat the problem of energy level identification. For highly-ionized ions we find it convenient to map levels in the following fashion.

The energy levels in each of the two structure calculations must first be split into groups according to the parity and total momentum of the states. This is readily accomplished using the functions developed to construct averaging groups as described above. Within a single such group we energy order the levels in the two calculations and then pair the two sets of levels together. There are two principle assumptions in this approach: the underlying physical model in the two calculations are similar, and the two calculations are of identical level sets. In many cases, one calculation may contain many more levels than another (generally in the form of additional levels with higher-lying valence electrons). This complication may be resolved by further defining the "groups" to include the shell electron occupation numbers. For highly-ionized atoms, we frequently find the shell occupation numbers to be good quantum numbers, even for calculations employing multi-configurational basis states. In particular

situations it may be found that some other scheme must be used to define the level "groups". We see that this represents a very minor change in the process – the approach itself is general.

Evaluation of kinetics models

Simple routines have been provided to solve the system of rate equations defined by the atomic model. These routines return a data structure which contains the solution to the rate equations, *i.e.* the level populations, and additional quantities of interest. This includes, for example, the column sum of the rate matrix (which should be identically zero in the steady-state). Quantities are extracted using the standard access routine for opaque data structures:

```
GetProperty(solnStruct, POPULATIONS, &pops);
```

which here requests the array of level populations.

The rate equations may be solved either in steady-state, or with time dependence. Further, the plasma state can be specified quite generally by an electron energy distribution and photon spectrum. For steady-state, constraints on the relative ion abundances may be applied.

Kinetics models in external physics application codes

A standard UNIX library exists to allow a user to link the evaluation and solving routines into their stand-alone physics code. The most direct way to get the kinetics models into an external code is, from within the shell, to write the model into a structured binary file. The model is then accessed from a user's FORTRAN or C code by reading the file (with the structured binary I/O library) and passing the resulting unit number to the routines which evaluate the rates for a specified plasma state. The resulting rate data structure is next passed to the solver routine, which solves the rate equations and returns the solution data structure. As within the shell, the numerical quantities of interest are obtained by use of the GetProperty() function. The full model-solving capabilities are available in this fashion: time-dependence, general plasma state specification, *etc.*

Graphical Displays

Given that a kinetics model may be constructed we must then consider means of visualizing it. A graphical representation in the form of a Grotrian diagram is a common format and so has been provided within the shell. In these diagrams, the energy levels are drawn as functions of some quantity which aids the user in evaluating the general level organization or in visualizing the transition processes in the model. The interconnections of levels are indicated in the form of arrows showing the processes or population fluxes of relevance. The independent variable and the processes shown depend on the interests of the user and the specific problem under study.

We have therefore chosen to provide a general Grotrian diagram capability which allows the user to indicate which levels to display and the horizontal offsets to be used. This capability follows the usage conventions of the more general Cartesian plot routines. In the Grotrian diagrams the graphic properties of the levels may be individually set (*e.g.* line thickness, color, *etc.*). Transition processes may also be shown and modified on an individual basis. The horizontal shift given to the levels may be arbitrarily specified, or based on properties of the energy levels themselves. For example, to shift levels according to their total momenta, the momenta is simply extracted from the model data structure (in the fashion already described) and then supplied to the plot routine as the new offset. It is equally convenient to offset levels by the number of electrons in the atomic state or by any other property of the levels. The shell functions used to average energy levels by classifying them into specific groups may also be used here to specify a more complicated scheme for the level offsets. This may even be done dynamically; we can initially construct a diagram with shifts determined by bound electron count and then reset the offset property to view the same levels offset by electron count *and* total momentum. When the diagram is shifted like this, any indicated transitions are automatically adjusted to continue connecting the appropriate levels. In addition to the function call interface, an interactive GUI editor is provided. A final point: since the full model is known to the plotter, it is possible to bring up a display of

information about an energy level once it has been "selected". This is available through the plot editor.

8. EXAMPLE II: AN EXPERIMENTAL DIAGNOSTIC

One aim of this project is to simplify the use of large-scale data sets in the study of plasma spectroscopy. An example which illustrates the manipulation of massive amounts of data is to model the absorption spectrum of keV x-rays by a sample containing aluminum as a diagnostic element. Theoretical LTE absorption models are quite sensitive to the plasma temperature so that this technique can be used as a temperature diagnostic, given adequate knowledge of the density.[2] Several such experiments have recently been carried out with such high quality that the plasma temperature is constrained to within a few eV.[60,61]

The diagnostic technique depends on the use of a spectrometer of moderate to high resolving power ($\lambda/\Delta\lambda$ of 800 - 2000) to measure the Kα absorption by an aluminum plasma of a broad-spectrum backlighter. The experimental measurement covers the spectroscopically rich line region from 1520 - 1600 eV. The present example will reproduce the analysis of an earlier study[2] to illustrate the utility of the shell. A description of the experimental method may be found in the recent works of Perry,[62] or Koch.[63] We will here be concerned only with developing the aluminum temperature diagnostic. The method has recently been successfully extended to other elements in differing wavelength regions.[64,65] The diagnostic capability illustrated here can readily be used in these cases by a change in the underlying data file.

For a uniform slab, x-rays are exponentially attenuated according to their absorption coefficient, $\kappa_\nu(\rho, T)$, which is a function of density, ρ, temperature, T, and photon frequency, ν. The transmission is given by

$$T_\nu = I_\nu / I^\circ_\nu = \exp[-\rho L \kappa_\nu(\rho, T)],$$

where I°_ν is the direct backlight spectrum and I_ν is the transmitted spectrum. The quantity ρL is the areal mass density. A reasonably

accurate (± 20%) estimate of ρ, needed in the evaluation of κ_ν, has been obtained from side-on radiography. We assume that all experimental corrections to the data have been applied. To compare with experiment we must include the response function of the spectrometer, and so compute a quantity, T^I_ν,

$$T^I_\nu = \int R^I(\nu, \nu'; \Gamma)\ T_{\nu'}\ d\nu',$$

where $\Gamma = \lambda/\Delta\lambda$ is the resolving power of the instrument and $R^I(\nu, \nu'; \Gamma)$, the instrument function, is typically represented as a Gaussian function whose width is determined by Γ.

The calculation of T^I_ν depends ultimately on the existence of a highly-complete, spectroscopically accurate database of aluminum x-ray absorption coefficients.[2] The computation of the absorption coefficients requires substantial effort. That is, the atomic physics quantities, *e.g.* line positions and oscillator strengths, require high precision and represent a large collection of data since absorption by satellite lines (from multiple stages of ionization) is significant. The opacity calculation using these atomic data must be computed on a dense grid in ρ, T, and ν. Thus, the final database approached 100 megabytes in size. The development of the database is, in itself, a substantial theoretical effort.

The iterative modeling cycle is straightforward, but tedious in execution. The intent of this section is to indicate how the shell environment may be used to first make the underlying database widely available, and second, to make its use relatively straightforward, even for a novice. We begin by considering the ideal implementation. In principle the only activity required of the user is to select values for (ρ, T) and $\lambda/\Delta\lambda$. As these are varied we must re-evaluate those quantities which depend on them. The interface must allow a means of selecting ρ, T, and $\lambda/\Delta\lambda$, and for plotting the theoretical T^I_ν together with the experimental quantity. The ideal approach is a graphical user interface. Then, as the independent variables change, a recalculation of the appropriate dependent quantities is invoked by a specification of callback routines. The internal shell mechanisms thus ensure that all quantities are updated in proper sequence. For example,

changing ρ or T invokes a new interpolation in the opacity table. This change in κ_ν invokes a recalculation of T_ν, which results in a new T^I_ν, and finally in the plot itself being updated.

We now describe how we access the aluminum opacity database. This data was originally available as numerous text files, each of which contained a tabulation of κ_ν for a single value of (ρ, T). The chosen database implementation was to rewrite these files as structured binary data files and to use the existing distributed file server mechanism for access. Thus, we have a system that can access any of these files, from any machine on the Internet, simply by specifying the host archive machine and directory path. A loop loads the full opacity set into a data structure designed for interpolation in two independent variables. A companion interpolation function is used to evaluate $\kappa_\nu(\rho, T)$ as an arbitrary function of its arguments.

Figure 1 illustrates the complete interface as presented to the user. At this point, the opacity data set has already been loaded, default values of (ρ, T) and $\lambda/\Delta\lambda$ have been set and the dependent quantities of κ_ν and T^I_ν have been plotted. The shell graphical interface tools include simple mechanisms for setting one and two dimensional quantities with a click of a mouse button. We employ one of each of these in the figure. The plasma conditions are specified by clicking in the window labeled "Plasma State Picker"; $\lambda/\Delta\lambda$ is set with the slide control labeled "Resolving Power". In the transmission plot, we see the experimental quantity in dashed pattern, the theory is represented as a solid line. The sequence of figures which follow show the entire analysis of the experiment as carried out with three clicks of a mouse button.

Initially (Figure 1) we observe from the transmission spectrum that the computed charge state distribution represents a state that, compared to experiment, is overly ionized. For reference, we note that the main lithium-like absorption occurs at about 1580 eV. By clicking in the state picker window we lower the temperature leading to the result in Figure 2. Now, while the overall ionization balance matches the experiment, we note that we need to increase the resolving power. This is clear from the

lack of detail in the theoretical spectrum. Doing so leads to Figure 3, which is an acceptable comparison. Now iterate on the density, which is far less sensitive, to bring it into line with the experimentally determined value (0.0257 ± 0.005 g/cm^3). Since decreasing the density results in further ionization of the plasma, we compensate by simultaneously lowering the temperature. We thus arrive at the final result, consistent with all experimental constraints, as shown in Figure 4.

9. SUMMARY

A user-oriented plasma modeling environment has been described. This environment has been developed with great attention to facilitating the data management and manipulation activities encountered in the design and analysis of plasma spectroscopy experiments. The individual components of the project, the interpretive language processor, database communications, graphical display and interface, have been described in detail. Some effort has been made to indicate how these different components interact to provide further capability. Examples have been presented to show the potential of this approach in settings of general interest.

The first example, the construction of atomic kinetics models, demonstrated the utility of the interactive shell language in developing large, correlated physics data sets. The selection of the model's level structure, beginning with the database and continuing into the shell, has been discussed in considerable detail to demonstrate the flexibility and power of the approach. Atomic kinetics models can now be developed in a straightforward, uncomplicated fashion. The approach is sufficiently general that a user is free to develop level averaging schemes of his own design, as appropriate to his application.

The second example was to develop a graphical user interface to eliminate the rote activities involved in spectrum-matching data analysis. The analysis package was readily constructed – a consequence of the general approach taken in designing the shell and its underlying code subsystems. This example additionally validated the use of structured binary data files

and data access over a network in a physics data analysis setting. With a change in the absorption coefficient database, this analysis capability may be applied to transmission experiments involving other elements and transition arrays.[64,65] We note also the potential for application to experiments and data sets involving Stark-broadened line profiles.[66]

Additional work will be carried out to move the shell from a research to a general user environment. It is essential also to establish additional collaborations within the plasma modeling community to ensure the generality and flexibility of the implementation. We will be soliciting interest from reseachers, in both experiment and theory, to develop additional applications.

This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

1. J. Abdallah Jr., R. E. H. Clark, and J. M. Peek, *Phys. Rev. A* **45**, 3980 (1992).

2. C. A. Iglesias, J. K. Nash, M. H. Chen, and F. J. Rogers, *JQSRT* **51**, 125 (1994).

3. R. C. Mancini, C. F. Hooper Jr., and R. L. Coldwell, *JQSRT* **51**, 201 (1994).

4. C. A. Back, J. I. Castor, R. I. Klein, P. G. Dykema, and R. W. Lee, *Phys. Rev. A* **44**, 6743 (1991).

5. G. D. Pollak, N. D. Delamater, J. K. Nash, and B. A. Hammel, *JQSRT* **51**, 303 (1994).

6. J. J. MacFarlane, P. Wang, J. Bailey, T. A. Mehlhorn, R. J. Dukart, and R. C. Mancini, *Phys. Rev. E* **47**, 2748 (1993).

7. P. F. Dubois, *Computers in Physics* **8**, 7122 (1994).

8. W. Cunto and C. Mendoza, *Rev. Mexicana Astron. Astrofis.* **23**, 107 (1992).

9. R. K. Janev, "ALADDIN On-line Database for Atomic, Molecular, Particle-Surface Interaction and Material Properties", International Atomic Energy Agency, Atomic and Molecular Data Unit, Vienna, Austria (1994).

10. W. C. Martin *et al.*, "NIST Laboratory Program on Atomic Spectroscopic Data for Astronomy", Astron. Soc. Pacific Conf. Series, in press (1995).

11. G. Fisher, *IEEE Transactions on Software Engineering* **14**, 774 (1988).

12. G. Eichhorn *et al.*, Astron. Soc. Pacific Conf. Series, in press (1995).

13. P. Dubois *et al.*, "The Basis System", unpublished report UCRL-MA-118543 parts 1-6, Lawrence Livermore National Laboratory, Livermore, California (1994).

14. D. H. Munro, "Yorick, an Approach to Scientific Computing", private communication, Lawrence Livermore National Laboratory, Livermore, California (1994).

15. "C-Lite Programmer's Reference Guide", unpublished report, Ellery Systems, Inc., Boulder, Colorado (1992).

16. S. A. Brown, M. Folk, G. Goucher and R. Rew, *Computers in Physics* **7**, 304 (1993).

17. R. K. Rew and G. P. Davis, "NetCDF: An Interface for Scientific Data Access", *IEEE Comput. Graph. Applications* **10**, July, 76 (1990).

18. S. A. Brown, D. Braddy, and J. E. Moura, "PDBLib User's Manual", unpublished report M-270 Rev. 3, Lawrence Livermore National Laboratory, Livermore , California (1994).

19. R. A. Hulse, "The ALADDIN Atomic Physics Database System", eds. Y. Kim and R. C. Elton, in AIP Conf. Proc. 206, 63 (1989).

20. W. Martin, "Sources of Atomic Spectroscopic Data for Astrophysics", eds. P. L. Smith and W. L. Wiese, in Atomic and Molecular Data for Space Astronomy, Springer-Verlag, 121 (1992).

21. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, "PVM3 User's Guide and Reference Manual", unpublished report ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, Tennesee (1994).

22. P. F. Dubois, *Computers in Physics* 8, 72 (1994).

23. R. W. Scheifler and J. Gettys, *ACM Transactions on Graphics* 5, 79 (1986).

24. D. Heller and P. M. Ferguson, Motif Programming Manual, O'Reilly and Associates, Inc., 1994.

25. P. Klingebiel, "Using the AthenaTools Plotter Widget Set", unpublished report, University of Paderborn, Paderborn, Germany (1992); and references therein.

26. J. Abdallah Jr., R. E. H. Clark, J. M. Peek, and C. J. Fontes, *JQSRT* **51**, 1 (1994).

27. I. P. Grant *et al.*, *Comput. Phys. Commun.* **21**, 207 (1980).

28. B. J. McKenzie *et al.*, *Comput. Phys. Commun.* **23**, 233 (1980).

29. M. H. Chen, *Phys. Rev. A* **31**, 1449 (1985).

30. M. H. Chen, *Phys. Rev. A* **35**, 4586 (1987).

31. D. H. Sampson *et al.*, *Phys. Rev. A* **40**, 604 (1989).

32. H. A. Bethe and E. E. Salpeter, Quantum Mechanics of One- and Two-Electron Atoms, Plenum Publishing Company, 262 (1977).

33. R. S. Walling, quantum-defect calculations of non-relativistic oscillator strengths, private communication.

34. D. H. Sampson *et al.*, *Phys. Rev. A* **40**, 604 (1989).

35. B. G. Wilson, private communication.

36. R. E. H. Clark *et al.*, *Ap. J. Suppl.* **49**, 545 (1982); and references therein.

37. J. D. Perez, *J. Appl. Phys.* **48**, 1969 (1977).

38. D. Salzmann and A. D. Krumbein, *J. Appl. Phys.* **49**, 3229 (1978).

39. M. J. Seaton, *Proc. Phys. Soc. London* **79**, 1105 (1962).

40. A. Burgess and H. P. Summers, *MNRAS* **174**, 345 (1976).

41. D. H. Sampson and H. L. Zhang, *Ap. J.* **335**, 516 (1988).

42. H. van Regemorter, *Ap. J.* **136**, 906 (1962).

43. H. L. Zhang *et al.*, *Phys. Rev. A* **40**, 616 (1989).

44. R. S. Walling and J. C. Weisheit, *Physics Reports* **162**, 1 (1988).

45. A. Burgess and M. J. Seaton, *MNRAS* **120**, 121 (1960).

46. E. B. Saloman, J. H. Hubble and J. H. Scofield, *Atomic Data Nucl. Data Tables* 38, 1 (1988).

47. M. J. Seaton, *Planet. Space Science* 12, 55 (1964).

48. Ya. B. Zel'dovich and Yu. P. Razier, Physics of Shock Waves and High Temperature Hydrodynamic Phenomena, vol. 1, Academic Press, 393 (1966).

49. R. F. Post, *Plasma Physics* 3, 273 (1961).

50. M. Gryzinski, *Phys. Rev.* 138A, 336 (1965).

51. G. Elwert, *Z. Naturforschg.* 7a, 432 (1952).

52. H. W. Drawin, *Z. Physik* 164, 513 (1961).

53. D. L. Moores, L. B. Golden and D. H. Sampson, *J. Phys. B* 13, 385 (1980).

54. L. B. Golden and D. H. Sampson, *J. Phys. B* 13, 2645 (1980).

55. L. B. Golden, D. H. Sampson and K. Omidvar, *J. Phys. B* 11, 3235 (1978).

56. L. B. Golden and D. H. Sampson, *J. Phys. B* 10, 2229 (1977).

57. O. Bely and P. Faucher, *Astron. and Astrophys.* 18, 487 (1972).

58. R. K. Landshoff and J. D. Perez, *Phys. Rev. A* 13, 1619 (1976).

59. L. B. Golden and D. H. Sampson, *J. Phys. B* 13, 2645 (1980); and references therein.

60. T. S. Perry, S. J. Davidson, F. J. D. Serduke, D. R. Bach, C. C. Smith, J. M. Foster, R. J. Doyas, R. A. Ward, C. A. Iglesias, F. J. Rogers, J. Abdallah, R. E. Stewart, J. D. Kilkenny, and R. W. Lee, *Phys. Rev. Lett.* 67, 3784 (1991).

61. P. T. Springer, T. S. Perry, D. F. Fields, W. H. Goldstein, B. G. Wilson, and R. E. Stewart, "Measurements and Models of the Opacity of Hot, Dense Plasma", eds. E. S. Marmar and J. L. Terry, in AIP Conf. Proc. 257, 78 (1991).

62. T. S. Perry *et al.*, *JQSRT* to be published in this issue.

63. J. A. Koch *et al.*, *JQSRT* to be published in this issue.

64. P. T. Springer, D. F. Fields, B. G. Wilson, J. K. Nash, W. H. Goldstein, C. A. Iglesias, F. J. Rogers, J. K. Swenson, M. H. Chen, A. Bar-Shalom, and R. E. Stewart, *JQSRT* **51**, 371 (1994).

65. T. S. Perry, K. S. Budil, R. Cauble, R. A. Ward, D. R. Bach, C. A. Iglesias, J. K. Nash, C. C. Smith, J. M. Foster, S. J. Davidson, F. J. D. Serduke, J. D. Kilkenny, and R. W. Lee, *JQSRT* to be published in this issue.

66. R. C. Mancini, C. F. Hooper, Jr., and R. L. Coldwell, *JQSRT* **51**, 201 (1994).

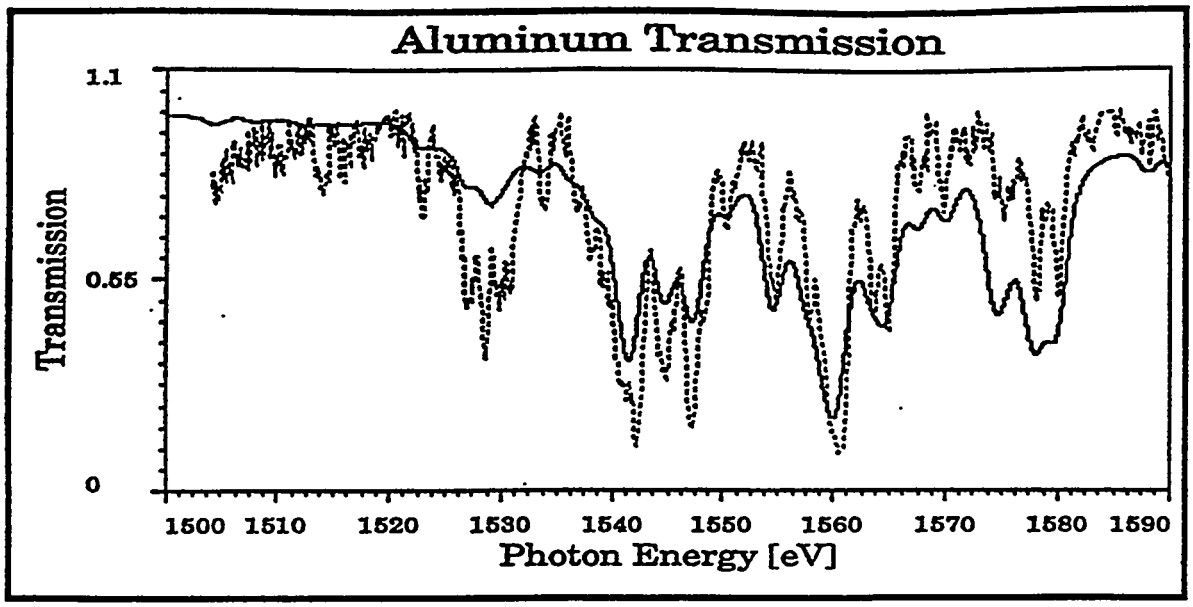
FIGURE CAPTIONS

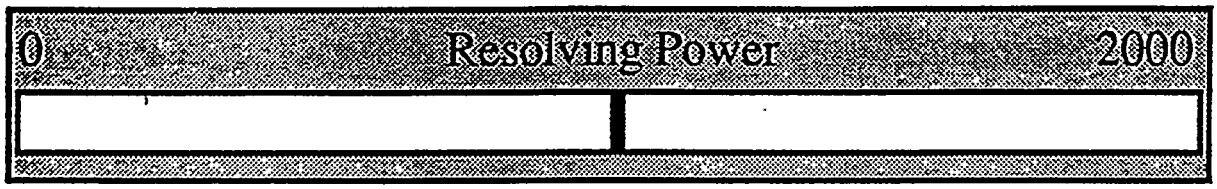
Figure 1. User interface as initially presented. In the transmission plot, experiment is dashed line pattern, theory is solid curve. Density scale in the "State Picker" is logarithmic. Judging from strength of lithium-like absorption in the 1580 eV region, the theoretical charge state distribution is overly-ionized compared to experiment.

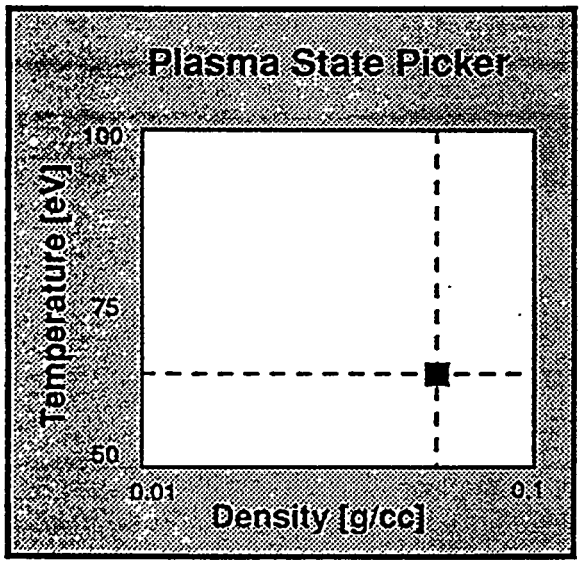
Figure 2. Selecting a lower temperature in the "State Picker" brings the theoretical ionization balance into agreement with the experiment.

Figure 3. The resolving power is increased to match that of the measurement.

Figure 4. A lower plasma density, consistent with radiographic measurement, is selected. The temperature is simultaneously decreased to maintain the ionization balance. Analysis now matches all experimental constraints.







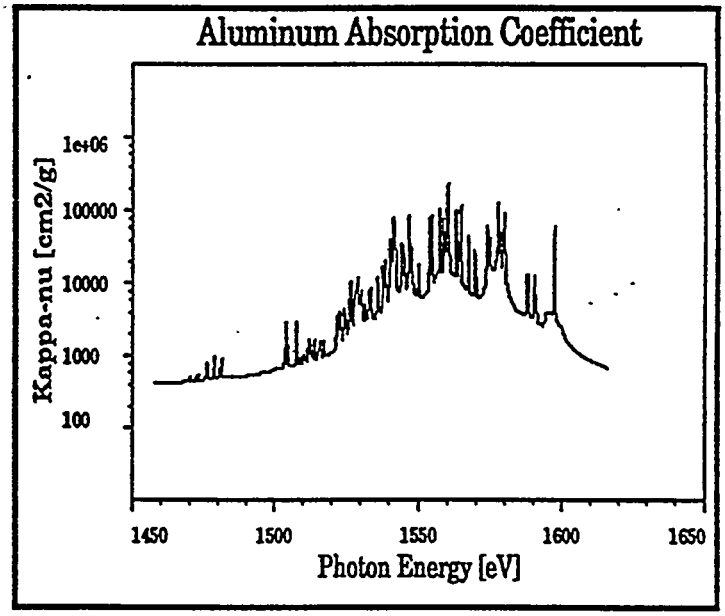
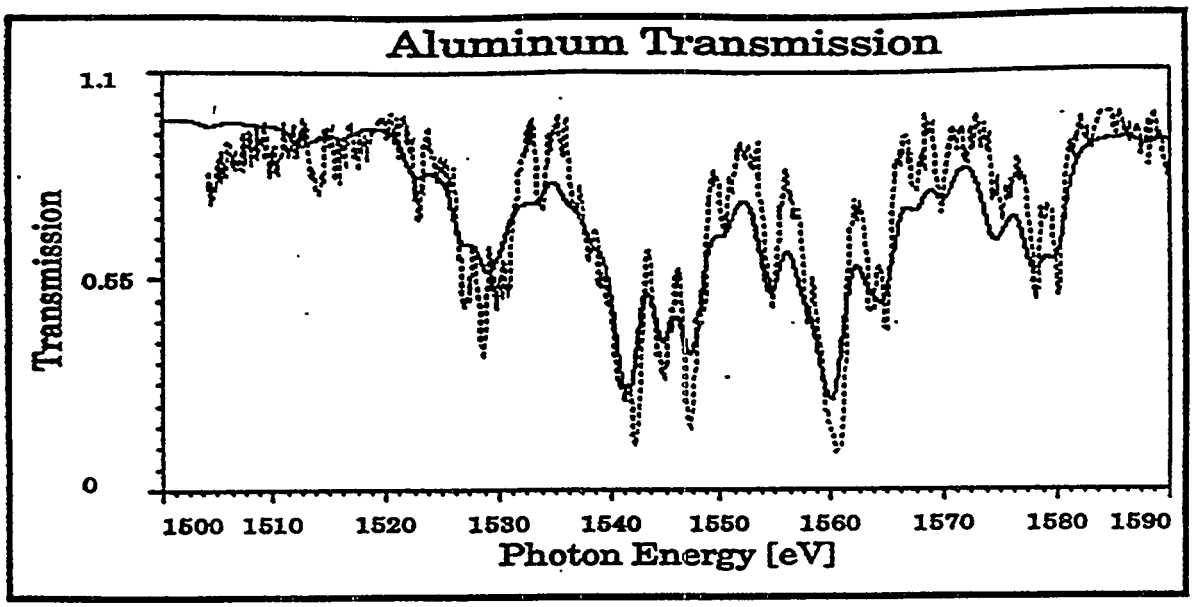
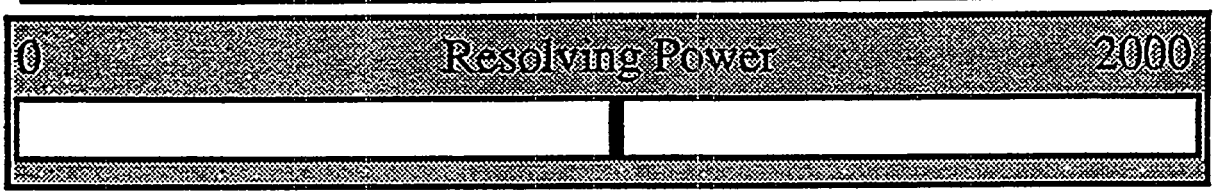
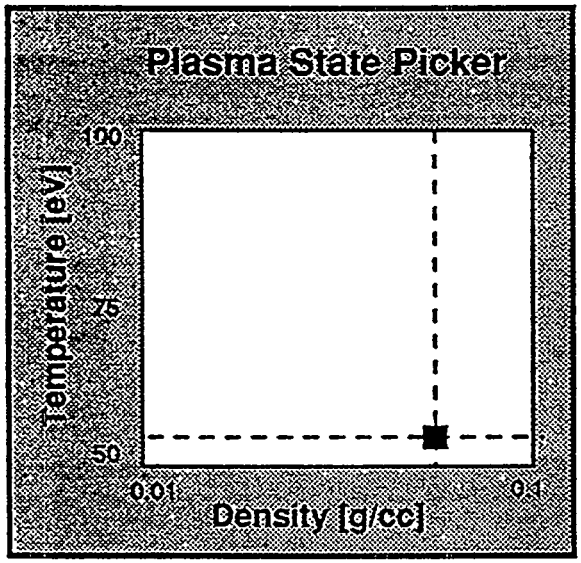


Figure 1 J.K. Nash







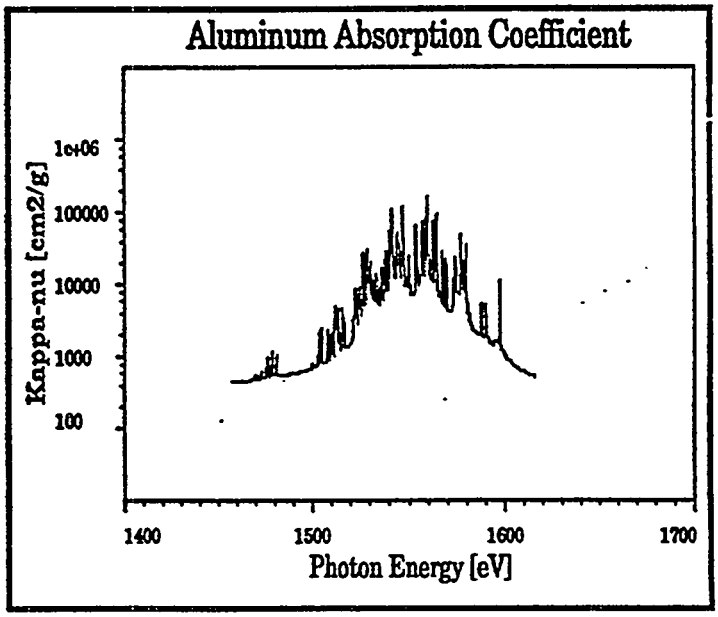
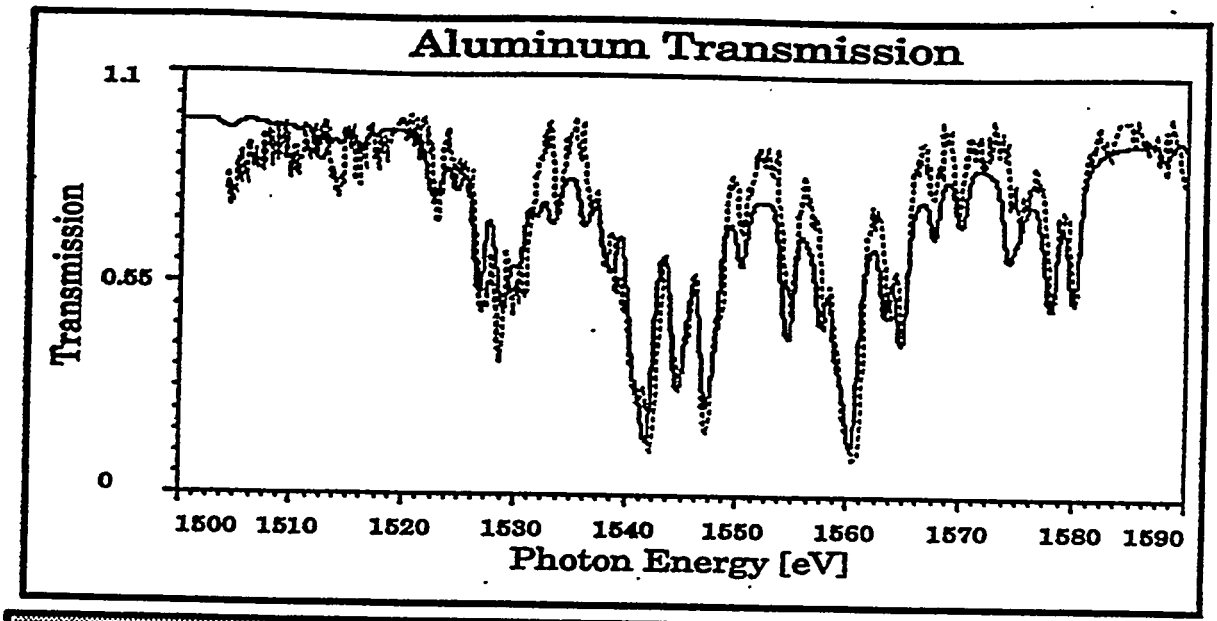
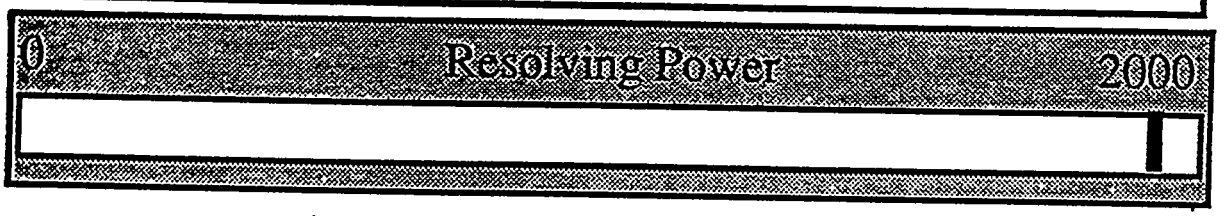
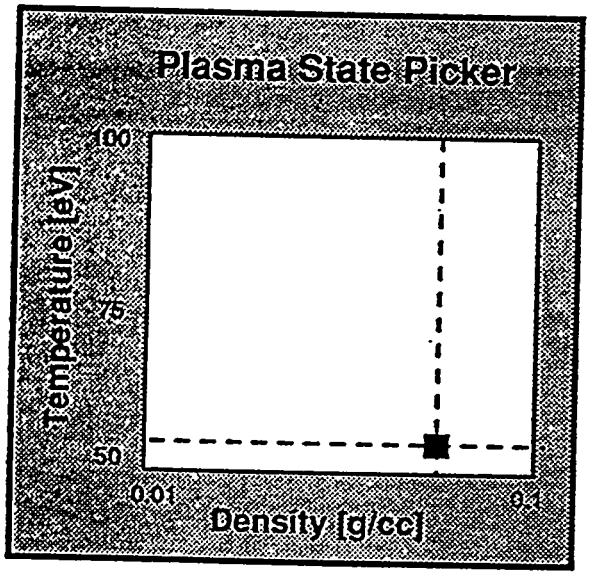


Figure 2 J. K. Nash







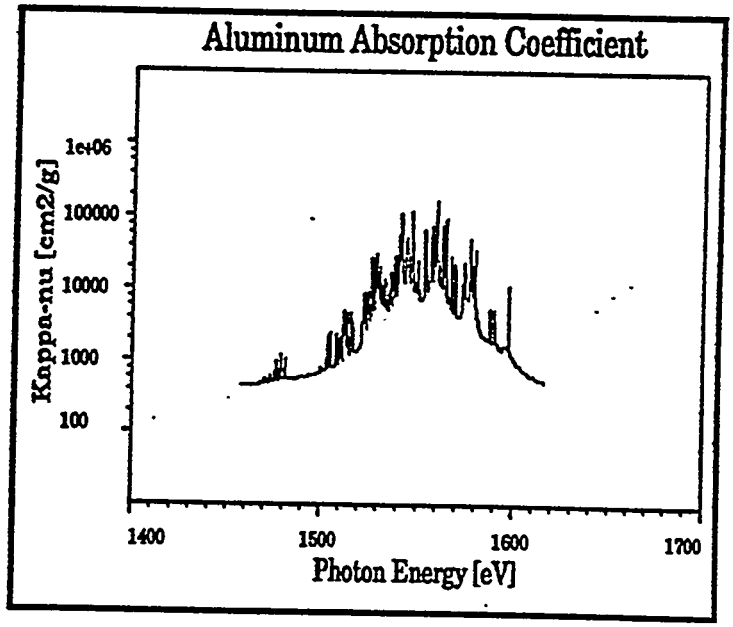
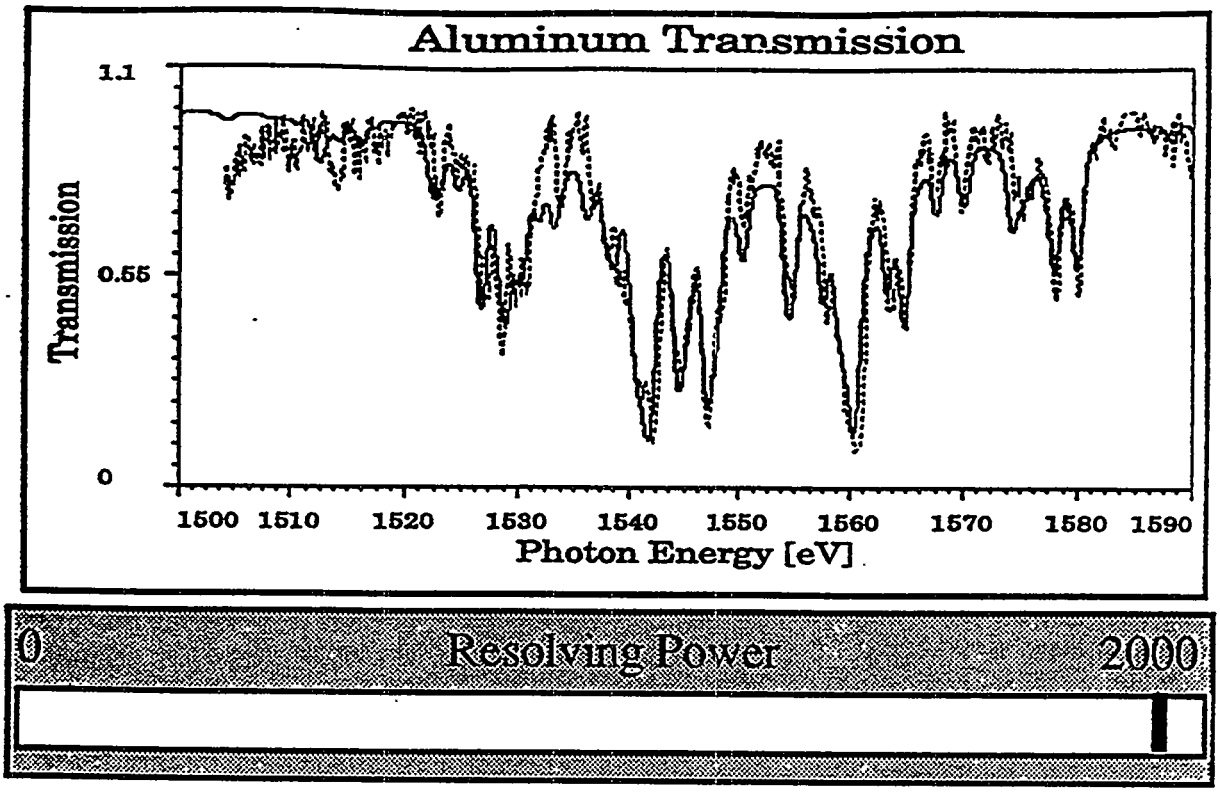
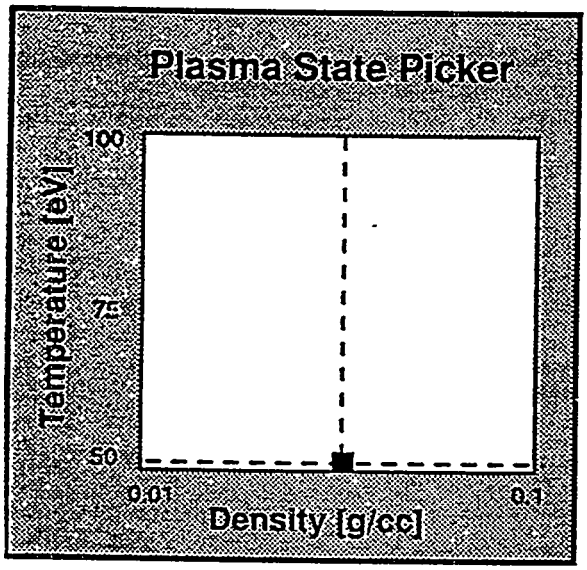


Figure 3 J.K. Nash





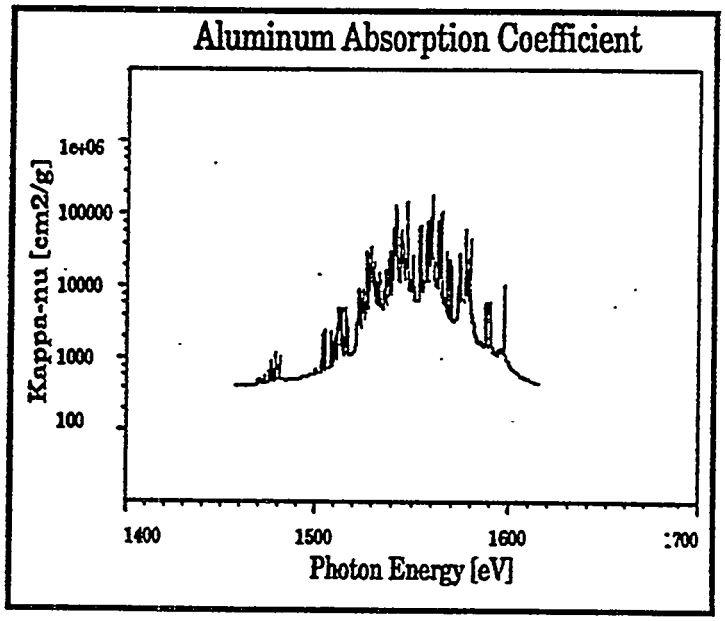


Figure 4 J. K. Nash